

图文 063、第9周答疑以及学员思考题总结汇总

599 人次阅读

2019-09-01 10:33:23

[详情](#) [评论](#)

狸猫技术

[进店逛](#)

第 9 周答疑以及学员思考题总结汇总！

问题：

老师，我这边发生第一次Full GC的时间是在老年代占满50M的时候，请问是为什么？还是说jstat打印出来的第一个FGC=1不算的。。我理了几个会触发FullGC的条件都不成立，第二次Full GC就能理解。

回答：

是的，第一次full gc是不太靠谱的，其实没太大必要，可以忽略掉他，关注第二次开始就好了

问题：

老师，stw的时候，系统停止，请求发生阻塞。那么老年代比如stw时间比较长，阻塞了很多请求。等这一次垃圾回收完之后，被阻塞的请求开始处理，又会创建很多对象，对jvm造成压力，然后老年代又要gc。所以stw时间久的话，会变相的给系统制造更高的并发，我的理解对吗？

回答：

也可以这么来理解，因为阻塞了很多请求，确实会造成瞬时处理请求过多

问题：

老师想请问下,2核4G，4核8G的linux机器，能够分配给堆内存的大小最大能有多少呢？

回答：

一般不能给到最大，操作系统和其他进程都要占用一些，比如4G的机器，给JVM个2G~3G，8G的机器，给JVM个4G左右，就差不多了

问题：

1.老师，g1对于大对象的判定规则是超过region的50%，有参数可以指定超过60%吗，还是通过参数指定region大小从而来指定大对象大小？

2.老师，这样理解对吗，parnew回收器和serial回收器的步骤是一样的，只不过parnew是并行的，g1的younggc也跟它们一样，不过它可以指定停顿时间，这中间是串行还是并行呢

相关频道



从 0 开
战高手
已更新1

回答：

- 1、g1有参数可以控制大对象的，但是建议不要改变
- 2、对的，你理解没错，其实都是类似的，多线程的话是并发回收的

问题：

像照片这类的图片我们一般都存在阿里云的OSS上，在数据库里只保存一个地址，这样的话个人主页只是文字和连接，最多也就1k吧，请问案例中为啥个人主页的数据会有几MB？

回答：

其实文章里解释了，个人主页除了图片还有大量文字，就是每天发的心情说说之类的东西

问题：

有一个有趣的现象，为什么 -XX:CMSFullGCsBeforeCompaction=5是大部分公司的设置呢？

我看了下我们公司也是设置为5，据我所知，这个参数模式是0的吧。设置为5这个“规则”是在那本书或者那个文章说出来，然后大家都遵守的吗？感觉是是不是误导了大家。

回答：

是的，是有人推荐设置为5，但是其实要考虑一下，如果通过优化之后，让full gc的频率很低，其实就完全可以让他每次full gc之后都compaction一次，那一次fullgc慢点而已，但是不至于大量内存碎片导致下一次fullgc提前到来

问题：

老师，访问量和PV这个我可以理解，但是QPS和TPS就区分不出来了。百度了一下，感觉说的都一样，太官方了，看不明白。您可以大白话详细说说QPS和TPS区别吗？用压测工具对一个接口压测，出来的是QPS还是TPS

回答：

压测工具的是QPS，就是你每秒多少个请求，TPS是每秒的事务量，这个一般用于数据库层面，就是数据库每秒多少个事务

问题：

测试服务器，1G的新生代，1G老年代，E区和S区6:2，项目启动的时候，进行了6次YGC,4次FGC，感觉太难优化了

回答：

启动就多次gc，一般就是启动的时候系统内置对象太多，只能考虑增加机器内存了，分配更大内存空间

问题：

如果用G1回收，它选择回收一部分region，会不会在老年代中对象块与块之间存在空缺部分（块1和块3之间的块2被回收了），下次在分配对象时发现块与块之间的空隙部分放不下新对象（这空隙部分算不算碎片）？

回答：

是选择一部分的region，不是选择region中的一部分对象来回收，所以g1是没有碎片问题的。任何一个region回收的时候，都是复制算法，他会把region里的存活对象拷贝到其他region区，然后直接清空这个region剩余的垃圾对象。

问题：

分析了下我们的系统，服务器不接受任何请求的情况下，大概16分钟左右1次YGC，1.5天左右1次FGC，每次YGC大概8M左右对象进入老年代。但是服务器每次启动都会有3-4次FGC，不知道怎么样去优化

回答：

服务器启动的时候很多内置对象，初始化之类的，那个不需要优化，核心是运行期间的优化

问题：

例子还是忽略了200M大小大于了S区百分之五十，从而下一次minor gc触发后，会直接把200M放入old的情况吧？

回答：

不是的，当时是举例子，你可以把Survivor认为是调整更大一些，不触发动态年龄判定

问题：

老师，QPS跟TPS的关系是什么？是不是可以说一个TPS可能包含对个QPS

回答：

其实一般系统都说QPS，TPS一般用于数据库里的概念，数据库层面每秒多少个事务

问题：

所以说高并发加慢处理是导致full gc的元凶之一，从而带来更多的性能问题

回答：

对的，其实后续大量的案例都是围绕这个中心点来展开的，反复用大量的案例把最根本的jvm优化原理强化到你们的大脑里去

问题：

老师，你有遇到过压测系统的时候，响应时间很慢，但是此时单独打开浏览器访问却很快的情况吗？这个是什么道理，请教下，谢谢

回答：

压测的时候慢是因为最高负载把你的机器资源打满了，包括网络和CPU，此时当然系统响应速度变慢了

问题：

请教一下 G1相对其他回收器有什么劣势吗，还是说很多地方不用G1只是因为没必要呢

回答：

G1未来会成为一个默认的垃圾回收器，好处就是只要指定一个垃圾回收的停顿时间，就可以让g1自动优化了，你一般不用过度优化。

坏处是你没法精准把控内存分配、使用、垃圾回收，所以有的时候优先使用cms+parnew，如果没问题就不一定用g1。

问题：

这些心情啥的我感觉也是是不是也应该分页展示，比如先加载十条，下拉再加载十条这样，感觉这样一次好像也就几k就够了

回答：很多时候他会一次性加载出来一大批，那也是够多的了

问题：

老师，大堆使用G1 但是设为了不影响系统性能，g1设置了回收允许停顿的时间，每次回收不多少内存，也就是腾出来的内存并不多，这样会不会提现不出大堆的优势

回答：

不会的，大堆你可以尽情的用，但是每次回收一小部分而已，保证垃圾回收不要影响太大

问题：

CMS不是扫描老年代的对象吗？那在重新标记之前进行一次YGC，YGC回收年轻代的对象，对提升重新标记的性能有所帮助吗？

回答：

CMS扫描老年代的对象是没错的，但是有的时候年轻代和老年代之间的对象有引用关系呢？是不是就会扫描到年轻代去了？

所以提前young gc一次，可以清理掉一些年轻代对象，是有助于提升CMS的重新标记阶段的性能的

问题：

不同的机器配置应该是给出不同的jvm参数的

1、JVM内存超过4G且对系统响应时间敏感的是不是应该采用G1？

2、对高并发，容易产生阻塞的系统，是不是考虑减小SurvivorRatio的值？这样可以给S区分配更大的空间，避免短命的对象进入老年代。这样可能会导致YGC会更频繁些，但YGC很快，关系不太大

回答：

1、没错，作为架构师或者team leader应该在团队内或者公司内推行统一的jvm参数模板

2、超过4G还不至于必须用G1，一般超过16G以上的机器可以考虑用G1，普通的机器都是2C4G，或者4C8G的，哪怕是CMS+ParNew也没问题的

3、高并发、大数据量的系统，建议是让系统负责人根据实际情况去优化各种参数，具体方法参加之前我们讲解的那套理论和思路即可

学员总结：

对并发高且核心的系统，还是应该采用之前文章讲授的方法：预估系统的并发量 ---选择合适的服务器配置 ---根据JVM内存模型---设定初始参数---对系统进行压测---观察高峰期对象的增长速率、GC频率、GC后的存活

然后根据实际的情况来设定jvm参数 --- 最后做好线上jvm监控。我能想到的就这么多，请老师补充

回答： 对的，核心系统一定是要走一整套思路来优化JVM参数的

问题：

学到这里我有个想法，就是把老师讲授的优化思路做成一个监控系统，系统通过jstat获取数据，然后给出jvm参数设置建议值，并给出这么设置的理由。不知道有没有这个必要？我感觉还挺好的

回答：

其实一般没必要，用一个统一的jvm参数模板就能解决80%的系统问题了，一般合理的参数可以让普通系统都没什么JVM的性能问题。但是如果要做一个类似的工具也没问题的

学员总结：

打卡。以前没有注意有个倒序的按钮，每次都要下拉好久，经过老师点评，今天进来就倒排，很方便。

今日思考题来说，目前没有通用设置JVM模板，基本都是默认参数，可能是用户量并不大的缘故吧。不过学习这个专栏以后，可以注意很多细节，学习的知识到下个公司就成了救火队长了，哈哈哈。

回答：

非常好，你其实可以根据你们公司各个系统的情况，去线上看一看，尽量站在架构师的角度给出一个较为合理的JVM参数模板，这样无论是对于实际的工作还是跳槽面试都是很有好处的

问题：

CMSScavengeBeforeRemark这个参数本意是希望在CMS GC remark之前做一次YGC，正常情况下其实是会做一次YGC的

这个参数的好处是如果YGC比较有效的话是能有效降低remark的时间长度，可以简单理解为如果大部分新生代的对象被回收了，那作为根的部分少了，从而提高了remark的效率

回答： 对的，理解正确

问题：

动态年龄判断是超过了Survivor区域的50%。请问这区域是指两个Survivor的50%，也就是一个Survivor的内存容量还是一个Survivor的50%？

回答： 是指代的一个Survivor区域的50%

问题：

可能年轻代的某个 GC Root，它引用了老年代的某个对象，这个对象就不能清除，所以CMS应该也要扫描年轻代GC Root，再进行一次YGC就可以减少扫描的年轻代GC链路。

另外G1基于Region收集，通过RememberSet记录引用关系来避免全堆扫描。不知道我解答的对不对，老师看到的话麻烦补充下。

回答： 解答的基本没什么问题，很好

学员总结：

到了这里和以前的零散经验链接起来了。由于CMS remark阶段需要扫描新生代(原因太长，不说了)，所以整个堆中数量会影响remark阶段的耗时，所以remark之前添加一次可中断的并发预清理(其实就是继续执行并发标记)

另外为了防止并发预清理阶段等太久都不发生young gc，提供了CMSMaxAbortablePrecleanTime 参数来设置等待多久没有等到young gc就强制remark。默认是5s。

但是最终一劳永逸的办法是，添加参数CMSScavengeBeforeRemark，让remark之前强制Minor GC

回答： 是的，总结的非常好

问题：

为什么重标记的时候提前做一次young gc会提高效率？重标记不是只针对老年代的对象进行标记的吗？就算young gc减少了部分对象，重标记也不会去新生代里查找的呀

回答：

老年代扫描的时候要确认老年代里哪些对象是存活的，这个时候必然会扫描到年轻代，因为有些年轻代的对象可能引用了老年代的对象，所以提前做young gc可以把年轻代里一些对象回收掉，减少了扫描的时间，可以提升性能

问题：

老师看到这里我又产生很多问题

1.文章中提到大量metadata的软引用经过young gc就回收了。之前我们都很清楚young gc,old gc的触发条件，那么metadata什么时候触发呢？

2.文中的案例中，虽然反射会不断的创建各种奇怪的类，难道每次调反射都会创建不一样的奇怪类吗？如果不是的话，他直接引用已经创建过的奇怪类，也不至于内存一直暴涨啊。

3.这是一个比较非常规的案例，老师对于这种非常规的情况，我们的分析思路与之前要有所改变吗？

回答：

1、metadata区域的gc就是他自己满了就触发

2、从jdk底层实现机制而言，他会创建各种不同的类

3、其实非常规案例，你们也得了解一下，但是排查思路没什么变化，看文章里的思路，其实还是一步一步去排查的

问题：

请问老师，每次Full GC都会回收永久代么？还是说永久代满触发的FullGC才会回收永久代？1.8前的永久代和1.8的Metadata区域的回收时机是一样的么？

回答：

无论是old区满了触发的full gc，还是metadata区域满了触发的full gc，都会执行metadata区域的gc

问题：

老师，动态生成的类是什么类加载器加载的呀？是系统类加载器吗，是的话，那之前不是说一个class的回收需要加载他的类加载器已经回收了，这个时候系统类加载器是回收了吗？

回答： 对的，已经回收了

问题：

打卡。今天这个案例应该是反射生产的软引用类在触发young gc时实例对象被立刻回收了，而方法区的类对象还在，下次再反射调用的时又得重新加载这个类对象并在年轻代实例化，这样就导致方法区同样的类对象越来越多，很快就需要进行垃圾回收。是这样吗？

如果不是这样那方法区为什么会一直在增长？这些增长的类对象都是不同反射调用的产生的吗？好多疑问。

回答： 感觉你可能还是没完全理解透彻，可以把本文多看两遍，他其实讲的是对类的回收，类自己本身就是一个Class对象

问题：

打卡 本篇内容应该讲的是因为设置了这个参数，导致young gc时直接回收了大部分反射时jvm创建的软引用对象，导致下一次调用反射继续创建类 和class。而class被放在元空间，导致元空间很快就满了，接着就是full gc

回答： 对的，你理解的完全正确

学员实践总结：

非常感谢老师的回复。排查了survivor 区放不下的原因，并把 survivor 加大了内存，然后从新捋了一下问题

项目刚启动之后对象都在 Eden区(很大一部分是永久存活的对象), 压测之后永久存活的对象达到条件进入老年代了

老年代我分了20M, 发生第一次FullGC 之后老年代基本满了(基本都是永久存活的对象, 快放不下的感觉)

后面继续触发所有的 MinorGC 时, 符合条件的: 老年代可用空间小于历次进入老年代总对象平均大小, 所以导致一直是 FullGC

针对这个问题, 我把老年代设置成了 100M(足够存永久存活的对象了), 继续压力测试, 对象正常走 E-S0-S1 了, 使用 jstat 看情况, 感觉没有10天半个月都不会触发 fullgc 了。

回答: 非常好, 活学活用, 直接就优化上自己的系统了

问题:

老师, 这个系统运行逻辑是这样吗。比如我点击某个用户的个人主页, 如果在redis缓存中存在, 则直接返回, 如果不存在, 则去数据库请求, 这个过程会创建一系列对象, 多达几m, 然后放到redis缓存去。

由于方法生成这些对象后出栈, 从而让堆对象成了垃圾对象, 以此类推来塞满堆从而触发gc。这样理解对吗老师

回答:

是的, 就是你理解的这个样子

问题:

这篇看了几遍还是比较疑惑:

- 1、正常加载的class与文章所说通过反射加载的class, 全部都是软引用吗?
- 2、“奇怪的Class对象” 是指XXX.class? 还是类似GeneratedSerializationConstructorAccessor.class?
- 3、Young GC为什么也会带着回收metaspace里的class对象?
- 4、即使gc后反复重新创建, 不是应该只创建不存在的class对象吗? 也不会超过最开始的100啊? 难道会重复创建相同的class对象?

回答:

- 1、这里仅仅说的是JVM自己生成的Class是软引用
- 2、是指那个很长名字的类, 其实除了那个类以外还有别的类似的生成的奇怪的类
- 3、GC的时候会去检查软引用这种特殊的对象, 如果满足条件就会回收, 那些Class都是软引用的
- 4、对的, 因为JDK源码里的实现有一些问题, 所以导致并发环境下会重复创建一些Class

问题:

老师, 这个例子是不是理解下来就是对full gc由metaspace触发的场景讲解。使得我们明白full gc触发不仅仅只是堆空间的old区域不足导致。而原理都是一样, 都是对应的区域放满了, 放不下了新的对象触发full gc。只是这里的情况是反射导致的对class对象软引用是引用的在非堆的方法区?

如理解有误, 望老师指出, 指正, 谢谢

回答: 对的, 你理解的没错

问题：

老师好，后面自己试了下，补了下课。简单来说是不是可以理解为，对于反射的大量，多次调用，会因为nativemethodaccessor的次数影响，默认15次，从而生成最终的generateXXXaccessorXXX的类

这个类的作用在于反射的方法调用转化为本地调用，提升性能。但对于该类的method方法的软引用被回收了，所以导致元数据区多次生成相同的类。导致full gc。感觉这块讲不清楚了。。。

回答：其实你理解的没问题，就是这个意思

学员总结：

初始标记：标记由4种gc roots直接关键的对象。

并发标记：对老年代所有的对象进行trace，看是否能与gc roots建立间接关系，我就是gc roots是否可达。

最终标记：标记并发标记阶段引用变动的对象。

并发清理：并发清理掉可回收的内存，但是因为用户线程依旧在运行，所以每次full gc都会清理不干净，产生浮动垃圾。

回答：总结的很好

问题：

你好，软引用对象的回收，为什么会导致下一次调用反射会继续创建类呢，这里的class放在元空间是说的class还是class对象？

回答：Class就是Class对象，具体为什么会因为软引用回收之后继续有新的Class生成，其实是JDK内部本身的一个缺陷，他会在这个场景下生成很多的Class。

具体JDK源码我没分析，因为那个太繁琐，大家其实在这里记住反射场景下的这个问题即可

问题：

请问一下dmp打印的内容怎么分析,我用jhat命令查看得到的是下面这样的结果,对象后面只有一个地址,没有显示对象大小,点进去看一个对象的详情也没有显示对象的大小

而且新生代和老年代的对象都混在一起, 那从哪些信息可以判断老年代的哪个对象占用的空间最大

回答：jhat是显示出来对象数量和大小的，后面案例中我们会分析MAT、Visual VM之类的更好的工具，别着急

学员总结：

1.分析机器情况（机器配置，堆内存大小，运行时长，FullGC次数、时间，YoungGC次数、时间）

2.查看具体的jvm参数配置

3.然后根据JVM参数配置梳理出JVM模型，每个区间的大小是多少，画出来JVM模型（考虑每个设置在申请情况下会执行GC）

4.结合jstat查看的GC情况，在结合JVM模型进行二次分析


5.jmap dump内存快照，通过jhat或者Visual VM之类的工具查看具体的对象分类情况

6.根据分析的情况再具体到问题（Bug、或者参数设置等问题）

7.修复Bug, 优化JVM参数

回答: 总结的非常好

Copyright © 2015-2019 深圳小鹅网络技术有限公司 All Rights Reserved. 粤ICP备15020529号

 小鹅通提供技术支持